Assignment 15 Solutions

Q1 Given an array **arr[ ]** of size **N** having elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.Next greater element of an element in the array is the nearest element on the right which is greater than the current element.If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Example 1:

Input:

N = 4, arr[] = [1 3 2 4]

Output:3 4 4 -1

Explanation

In the array, the next larger element

to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ?

since it doesn't exist, it is -1.

Example 2:

Input:

N = 5, arr[] [6 8 0 1 3]

Output: 8 -1 1 3 -1

Explanation:

In the array, the next larger element to

6 is 8, for 8 there is no larger elements

hence it is -1, for 0 it is 1 , for 1 it

is 3 and then for 3 there is no larger

element on right and hence -1.

In [19]:
```
def findNextGreaterElements(arr):
    stack = []
    result = []

    for i in range(len(arr)-1, -1, -1):
        while stack and stack[-1] <= arr[i]:
            stack.pop()

        if not stack:
            result.append(-1)
        else:
            result.append(stack[-1])

        stack.append(arr[i])

    result.reverse()
    return result
```

In [20]:
```
arr = [1, 3, 2, 4]
result = findNextGreaterElements(arr)
print(result)

arr = [6, 8, 0, 1, 3]
result = findNextGreaterElements(arr)
print(result)
```

```
[3, 4, 4, -1]
[8, -1, 1, 3, -1]
```

Q2 Given an array **a** of integers of length **n**, find the nearest smaller number for

Q2 Given an array **a** of integers of length **n**, find the nearest smaller number for every element such that the smaller element is on left side.If no small element present on the left print -1.

Example 1:

Input: n = 3

a = {1, 6, 2}

Output: -1 1 1

Explaination:

There is no number at the

left of 1. Smaller number than 6 and 2 is 1.

Example 2:

Input: n = 6

a = {1, 5, 0, 3, 4, 5}

Output: -1 1 -1 0 3 4

Explaination:

Upto 3 it is easy to see

the smaller numbers. But for 4 the smaller

numbers are 1, 0 and 3. But among them 3

is closest. Similary for 5 it is 4.

In [21]:
```
def findNearestSmallerElements(arr):
    stack = []
    result = []

    for i in range(len(arr)):
        while stack and stack[-1] >= arr[i]:
            stack.pop()

        if not stack:
            result.append(-1)
        else:
            result.append(stack[-1])

        stack.append(arr[i])

    return result
```

In [22]:
```
arr = [1, 6, 2]
result = findNearestSmallerElements(arr)
print(result)

arr = [1, 5, 0, 3, 4, 5]
result = findNearestSmallerElements(arr)
print(result)
```

```
[-1, 1, 1]
[-1, 1, -1, 0, 3, 4]
```

Q3 Implement a Stack using two queues **q1** and **q2**.

Example 1:

Input:

push(2)

push(3)

pop()

push(4)

pop()

Output:3 4

Explanation:

push(2) the stack will be {2}

push(3) the stack will be {2 3}

pop() poped element will be 3 the

```
        stack will be {2}
```

push(4) the stack will be {2 4}

pop() poped element will be 4

Example 2:

Input:

push(2)

pop()

pop()

push(3)

Output:2 -1

In [23]:
```
class Stack:
    def __init__(self):
        self.q1 = []
        self.q2 = []

    def push(self, x):
        self.q1.append(x)

    def pop(self):
        if not self.q1:
            return -1

        while len(self.q1) > 1:
            self.q2.append(self.q1.pop(0))

        popped_element = self.q1.pop(0)

        self.q1, self.q2 = self.q2, self.q1

        return popped_element

    def top(self):
        if not self.q1:
            return -1

        while len(self.q1) > 1:
            self.q2.append(self.q1.pop(0))

        top_element = self.q1[0]

        self.q2.append(self.q1.pop(0))

        self.q1, self.q2 = self.q2, self.q1

        return top_element

    def is_empty(self):
        return len(self.q1) == 0
```

In [24]:
```
stack = Stack()
stack.push(2)
stack.push(3)
print(stack.pop())
stack.push(4)
print(stack.pop())

stack = Stack()
```

```
stack.push(2)
print(stack.pop())
print(stack.pop())
stack.push(3)
print(stack.top())
```

```
3
4
2
-1
3
```

Q4 You are given a stack **St**. You have to reverse the stack using recursion.

Example 1:

Input:St = {3,2,1,7,6}

Output:{6,7,1,2,3}

Example 2:

Input:St = {4,3,9,6}

Output:{6,9,3,4}

In [25]:
```
def insertAtBottom(St, top_element):
    if not St:
        St.append(top_element)
    else:
        popped_element = St.pop()
        insertAtBottom(St, top_element)
        St.append(popped_element)

def reverseStack(St):
    if len(St) <= 1:
        return
    else:
        top_element = St.pop()
        reverseStack(St)
        insertAtBottom(St, top_element)
```

In [26]:
```
St = [3, 2, 1, 7, 6]
reverseStack(St)
print(St)

St = [4, 3, 9, 6]
reverseStack(St)
print(St)
```

```
[6, 7, 1, 2, 3]
[6, 9, 3, 4]
```

Q5 You are given a string **S**, the task is to reverse the string using stack.

Example 1:

Input: S="GeeksforGeeks"

Output: skeeGrofskeeG

In [27]:
```
def reverseString(S):
    stack = []
    for char in S:
        stack.append(char)

    reversed_string = ""
    while stack:
        reversed_string += stack.pop()

    return reversed_string
```

In [28]:
```
S = "GeeksforGeeks"
reversed_str = reverseString(S)
print(reversed_str)
```

```
skeeGrofskeeG
```

Q6 Given string **S** representing a postfix expression, the task is to evaluate the

expression and find the final value. Operators will only include the basic arithmetic operators like *, /, + and -.

Example 1:

Input: S = "231*+9-"

Output: -4

Explanation:

After solving the given expression,

we have -4 as result.

Example 2:

Input: S = "123+*8-"

Output: -3

Explanation:

After solving the given postfix

expression, we have -3 as result.

In [29]:
```
def evaluatePostfix(S):
    stack = []

    for char in S:
        if char.isdigit():
            stack.append(int(char))
        else:
            operand2 = stack.pop()
            operand1 = stack.pop()

            if char == '*':
                result = operand1 * operand2
            elif char == '/':
                result = operand1 / operand2
            elif char == '+':
                result = operand1 + operand2
            elif char == '-':
                result = operand1 - operand2

            stack.append(result)

    return stack[0]
```

In [30]:
```
S = "231*+9-"
result = evaluatePostfix(S)
print(result)

S = "123+*8-"
result = evaluatePostfix(S)
print(result)
```

```
-4
-3
```

Q7 Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(int val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

You must implement a solution with `O(1)` time complexity for each function.

Example 1:

Input:

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output:[null,null,null,null,-3,null,0,-2]

Explanation:

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

In [31]:
```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, val):
        self.stack.append(val)
        if not self.min_stack or val <= self.min_stack[-1]:
            self.min_stack.append(val)

    def pop(self):
        if self.stack[-1] == self.min_stack[-1]:
            self.min_stack.pop()
        self.stack.pop()

    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.min_stack[-1]
```

In [32]:
```
minStack = MinStack()
minStack.push(-2)
minStack.push(0)
minStack.push(-3)
print(minStack.getMin())
minStack.pop()
print(minStack.top())
print(minStack.getMin())
```

```
-3
0
-2
```

Q8 Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:

In [33]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\rainwatertrap.png")
```

Out[33]:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. **Example 2:**

Input: height = [4,2,0,3,2,5]

Output: 9

In [34]:
```
def trap(height):
    if not height or len(height) < 3:
        return 0

    left = 0
    right = len(height) - 1
    left_max = 0
    right_max = 0
    water = 0

    while left < right:
        if height[left] < height[right]:
            if height[left] > left_max:
                left_max = height[left]
            else:
                water += left_max - height[left]
            left += 1
        else:
            if height[right] > right_max:
                right_max = height[right]
            else:
                water += right_max - height[right]
            right -= 1

    return water
```

In [35]:
```
height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]
print(trap(height))

height = [4, 2, 0, 3, 2, 5]
print(trap(height))
```

```
6
9
```

In [ ]: